Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays

Robert J. Francis, Jonathan Rose, Kevin Chung

Department of Electrical Engineering, University of Toronto, Ontario, Canada

Abstract

Field Programmable Gate Arrays are new devices that combine the versatility of a Gate Array with the user-programmability of a PAL. This paper describes an algorithm for technology mapping of combinational logic into Field Programmable Gate Arrays that use lookup table memories to realize combinational functions. It is difficult to map into lookup tables using previous techniques because a single lookup table can perform a large number of logic functions, and prior approaches require each function to be instantiated separately in a library. The new algorithm, implemented in a program called *Chortle* uses the fact that a K-input lookup table can implement *any* boolean function of K-inputs, and so does not require a library-based approach. Chortle takes advantage of this complete functionality to evaluate all possible decompositions of the input boolean network nodes. It can determine the optimal (in area) mapping for fanout-free trees of combinational logic. In comparisons with the MIS II technology mapper, on MCNC-89 Logic Synthesis benchmarks Chortle achieves superior results in significantly less time. [1]

1 Introduction

The Field Programmable Gate Array (FPGA) is a new approach to ASIC design that can dramatically reduce manufacturing turn around time and cost [Hsie88, ElGa89, Wong89, Ples89, Marr89, Bake90]. An FPGA consists of a regular array of programmable logic blocks that can be interconnected by a programmable routing network. The programmable nature of these devices requires new CAD algorithms to make effective use of the logic and routing resources.

This paper addresses the problem of technology mapping for FPGAs where the logic block implements combinational logic using lookup tables. A K-input lookup table is a digital memory containing 2^K bits of information that produces a single output boolean logic function of K or fewer input variables.

The motivation to study technology mapping for lookup table-based FPGAs is two-fold: First, the original FPGA [Hsie88] uses lookup tables and recent work [Rose89] suggests that lookup tables are an area-efficient choice for logic blocks. Secondly, conventional technology mapping algorithms [Keut87, Detj87, Lisa88] are unsuitable for mapping lookup tables because a separate library element is required to represent each of the logic functions that can be implemented by a K-input lookup table. A library containing the required 2^{2^K} elements is beyond the capability of previous techniques even for K=4.

We present a new algorithm, implemented in the *Chortle* program that avoids the library size problem by using the ability of a K-input lookup table to implement *any* sub-graph with K-inputs. Chortle divides the input boolean network into a forest of trees, and can efficiently determine the optimal mapping of each tree. A major feature of Chortle is that it considers all possible decompositions of every node in the network.

The Chortle program is compared to the MIS II technology mapper [Detj87] on several of the MCNC-89 logic synthesis benchmarks, for lookup tables with 2,3,4 and 5 inputs. For the smallest lookup table (K=2) the results are almost identical. For the larger lookup tables (K=4,5) the Chortle produces mappings with 4% to 28% fewer lookup tables than MIS II, typically in significantly less time.

The dynamic programming traversal of the tree used in the Chortle algorithm is similar to that presented in DAGON [Keut87] and MIS [Detj87]. A related work dealing with blocks that can implement more than one boolean function is the mapping for standard cell generators described in [Berk88]. Technology mapping of K-input lookup tables is different from the latter because a lookup table can perform *any* function of K inputs rather than a restricted class of functions such as And-Or-Inverts.

This paper is organized as follows. In Section 2 we define our technology mapping problem. In Section 3

[1] This work was supported by NSERC Operating Grant #URF0043298, a research grant from Bell-Northern Research and a research grant from ITRC.

the new algorithm is described. Section 4 compares Chortle to the MIS II technology mapper [Detj87] and describes how the library used by MIS is created.

2 Problem Definition

We assume that logic synthesis is divided into two steps: logic optimization and technology mapping. The input and output of the logic optimization step is a boolean network which represents a multi-input multi-output boolean function.

We use a directed acyclic graph (DAG) representation of a boolean network similar to that given in [Keut87]. Figure 1 illustrates such a representation of a boolean network. The boolean inputs of a network are represented by the input nodes of the graph, nodes a, b, c, d and e in Figure 1. We define an *input* node of a graph to be a node with no fanin nodes. A node n_f is a *fanin* node of n if there is a directed edge from the node n_f to the node n.

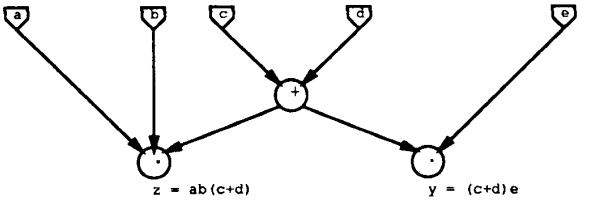

Figure 1: A Boolean Network

The boolean function represented by a non-input node is either the boolean operation AND or OR applied over the fanin boolean variables. Edges and nodes of the graph are labelled to indicate the polarity (inverted or non-inverted) of signals and to specify which nodes represent the output nodes of the boolean network.

Technology mapping for lookup table-based FPGAs takes an optimized network and produces a circuit of K-input lookup tables that implements the network. For all lookup tables in the circuit K is the same integer. The Chortle program minimizes a cost function consisting of the number of lookup tables, which is a measure of area.

A circuit of K-input lookup tables is represented by a cover of sub-DAGs on a DAG representing a boolean network satisfying the following conditions:

- Each sub-DAG in the cover must have only a single output node and K or fewer input nodes.
- Every edge in the network DAG must appear in one and only one sub-DAG in the cover.

The cover is a valid implementation of a network if for every output node in the network there is a corresponding output node of a sub-DAG in the cover that has the same boolean function.

We assume that the boolean network to be mapped has already gone through logic optimization. To retain the optimized network structure we impose the additional restriction that for every node in the network there must be at least one node in the set of sub-DAGs that has the same boolean function. Note that this restriction does not prohibit the duplication and decomposition of nodes.

Figure 2 illustrates a circuit of 3-input lookup tables that implements the boolean network specified in Figure 1. To simplify subsequent figures we will omit the edge and node labelling.

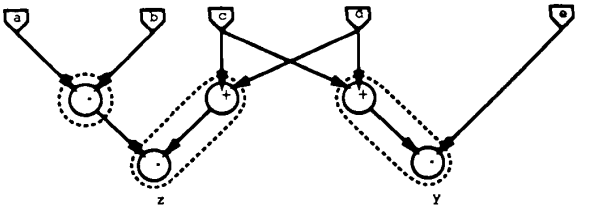

Figure 2: A 3-input Mapping

3 The Chortle Algorithm

The purpose of the algorithm is to find the minimum cost circuit of K-input lookup tables that implements an arbitrary boolean network represented by a graph G. The algorithm begins by converting the graph G into a forest of maximal fanout-free trees. Each of these trees is then mapped to find the minimum cost circuit that implements the tree. A circuit implementing the entire graph G is formed by combining the circuits that implement each tree in the forest.

Figure 3b illustrates how a forest of trees is created from the graph in Figure 3a. If a node n in the graph has out-degree greater than one then any edge, for example (n, a) in Figure 3a, originating from the node n is replaced by an edge, (n_a, a), originating from a new additional node. The new node node, n_a, is defined to have the same boolean function as the node n.

The following section shows how we find the optimal solution to the tree mapping problem.

3.1 Mapping a Tree

The heart of the tree mapping algorithm is a dynamic programming traversal of the tree which is outlined as pseudo code in Figure 4.

Starting with a given tree and a value for K the algorithm produces a minimum cost circuit of K-input lookup tables that implements the tree. The following are definitions of terms used in the pseudo code.

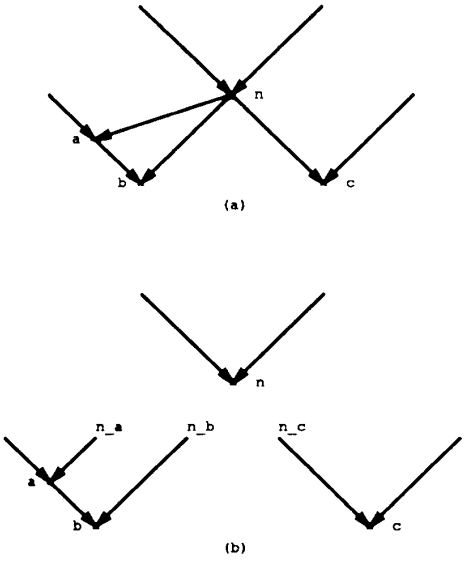

Figure 3: Creating a Forest of Trees

Definition 1 A *mapping* of node n, in a tree T, is a circuit of K-input lookup tables that implements the sub-tree of T that is rooted at n and extends to the leaf nodes of T.

Definition 2 The *root* lookup table of a mapping of the node n has as its single output the boolean function of the node n.

Definition 3 The *utilization* of a lookup table is the number of inputs, U, out of the K inputs that are actually used in a circuit.

The *utilization division* of a lookup table is defined in section 3.1.1.

In general, dynamic programming computes and records the solution to all sub-problems proceeding from the smallest to the largest sub-problem. Recording the solution to each sub-problem eliminates the need to recalculate it as part of the solution of any larger sub-problem. The sub-problem solved by Chortle is the computation of the minimum cost mapping of a node n, in a tree, where the root lookup table of the mapping has a given utilization U. The solution to this subproblem is denoted as $minMap(n, U)$.

In the following section we show that for any node n with fanin nodes $n_1 \ldots n_f$ if we have previously calculated $minMap(n_i, U_i)$, for all U_i from 2 to K, for every node n_i then we can calculate $minMap(n, U)$ for any value of U from 2 to K.

The entire tree is traversed in postorder starting from the leaf nodes and proceeding to the root node calculating $minMap(n, U)$, for all U from 2 to K, at every

```
Procedure MapTree(T, K) {

  For each node n in the tree T
  proceeding from the leaf nodes to the root node
  by postorder traversal {

    For utilization U = 2 to K

      CurrentBestCost = ∞
      CurrentBestMap = ∅

      For all utilization divisions, U of U {
        /* using dynamic programming */

        Construct a mapping M
        from the minimum cost mappings of
        the fanin nodes computed previously.

        /* M is the minimum cost mapping of n */
        /* where the root lookup table of M   */
        /* has utilization division U         */

        Calculate cost(M)

        if cost(M) < CurrentBestCost {
          CurrentBestCost = cost(M)
          CurrentBestMap = M
          }
        } /* next Utilization Division, U*/

      minMap(n, U) = CurrentBestMap
      } /* next Utilization, U */
    } /* next Node, n */

  Best Mapping of the tree T is minMap(n_root, K)
  }
```

Figure 4: Pseudo Code

node n in the tree. At the input (leaf) nodes of the tree we define the cost of $minMap(n_{leaf}, U)$ to be 0. It can be shown for any node n that:

$$cost(minMap(n, U)) \geq cost(minMap(n, K)), \forall U \leq K$$

Therefore by calculating $minMap(n_{root}, K)$ for the root node of the tree we have found the circuit with the fewest K-input lookup tables that implements the entire tree.

3.1.1 Calculating $minMap(n, U)$

For simplicity of description assume that the root lookup table of a mapping of a node n includes all the fanin edges of n. This implies that the fanin of n is less than or equal to K and that the node n is not decomposed. Section 3.1.3 considers the decomposition problem.

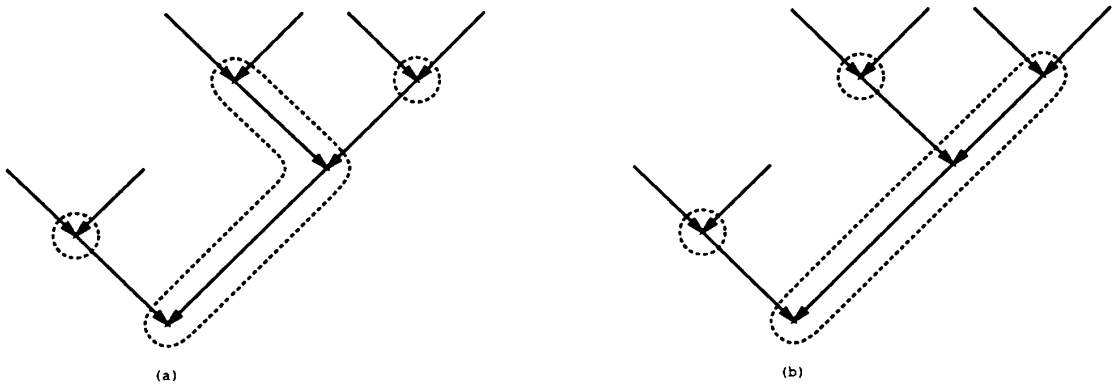

Figure 5: Utilization Divisions

If the node n has fanin nodes $n_1 \ldots n_f$ then the root lookup table of a mapping of n includes all the fanin edges of n and some sub-tree S_i rooted at each fanin node n_i.

We introduce the term *utilization division* to denote the distribution of the inputs to the root lookup table among these sub-trees. If u_i is the number of leaf nodes in the sub-tree S_i then the set $\mathcal{U} = \{u_1 \ldots u_f\}$ specifies the utilization division of the root lookup table because these leaf nodes correspond to the inputs to the root lookup table. Figure 5a illustrates a mapping using 4-input lookup tables where the root lookup table has utilization division $\mathcal{U} = \{1, 3\}$.

There may be many different mappings of a node that have the same utilization division of the root lookup table. Figure 5b illustrates a second mapping where the root lookup table has the same utilization division $\mathcal{U} = \{1, 3\}$.

There are many possible utilization divisions of the root lookup table of a mapping of a node. To find $minMap(n, U)$ for a given node n and utilization U we exhaustively search all possible utilization divisions $\mathcal{U}$ where $\sum u_i = U$. For each $\mathcal{U}$ we construct the minimum cost mapping of n where the utilization division of the root lookup table is specified by $\mathcal{U}$. One of these mappings will be $minMap(n, U)$. The following section describes how we construct these mappings.

3.1.2 Constructing A Mapping

The key to the dynamic programming approach is the construction of the minimum cost mapping of a node n where the root lookup table has a given utilization division $\mathcal{U} = \{u_1 \ldots u_f\}$. The desired mapping is constructed by combining a constructed root lookup table with the mappings $minMap(n_i, u_i)$ which have been previously computed for all fanin nodes n_i. For those fanin nodes, n_i, where $u_i = 1$ the mapping $minMap(n_i, K)$ must be used instead of $minMap(n_i, 1)$.

The constructed root lookup table includes all the fanin edges of n as well as a sub-tree S_i rooted at each fanin node n_i. For a fanin node n_i where $u_i \neq 1$ the sub-tree S_i is the root sub-DAG of $minMap(n_i, u_i)$. If $u_i = 1$ then the sub-tree S_i is simply the node n_i.

The root lookup table of the mapping $minMap(n_i, u_i)$ is eliminated from the constructed mapping if $u_i \neq 1$ because it is contained within the constructed root lookup table.

Figure 6 illustrates the construction of the minimum cost mapping of a node n with utilization division $\mathcal{U} = \{1, 3\}$ using 4-input lookup tables. The mappings selected for the fanin nodes a and b, $minMap(a, 4)$ and $minMap(b, 3)$, are shown in figure 6a. Figure 6b shows the constructed root lookup table which contains the root lookup table of $minMap(b, 3)$. The final mapping where the root lookup table of $minMap(b, 3)$ has been eliminated is shown in Figure 6c.

It is possible to prove that no other mapping of the node n where the root lookup table of the mapping has the utilization division specified by $\mathcal{U}$ can have fewer lookup tables than the mapping constructed by the above procedure [Fran 91].

3.1.3 Decomposition

So far we have assumed that there are no decompositions of nodes in our mapping of a tree. However, if the fanin of a node is greater than K then the node must be decomposed. By considering all possible decompositions of every node in the tree we may also be able to reduce the number of lookup tables required to implement the tree.

When a node n is decomposed intermediate nodes are introduced. Each intermediate node implements the boolean operation of n over some subset of the fanin nodes of n. The edges from this subset of fanin nodes

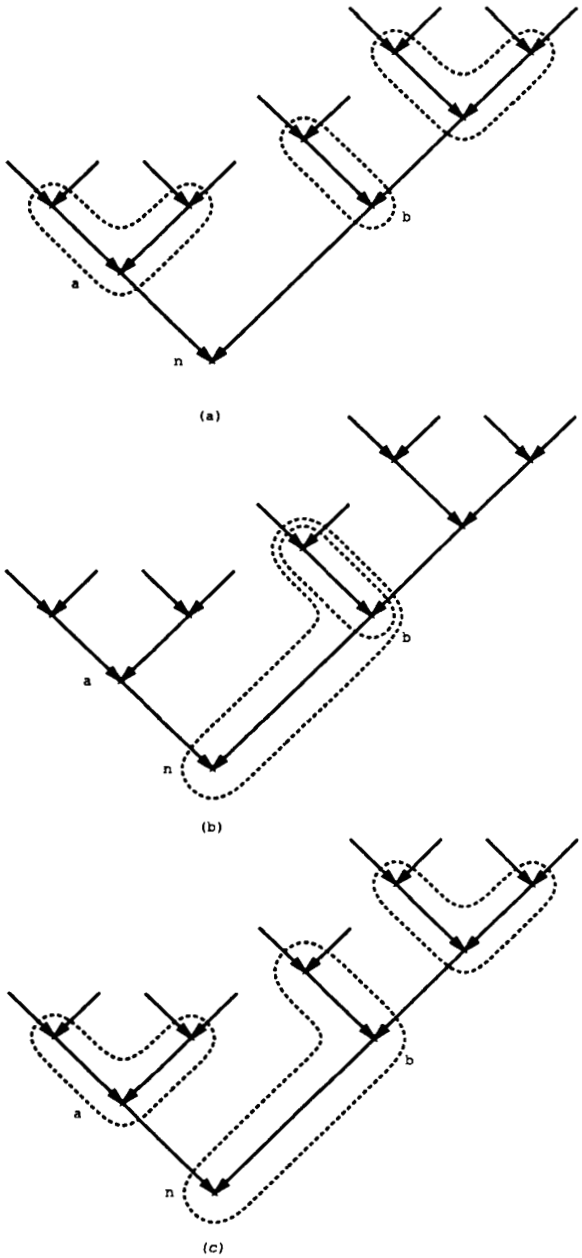

Figure 6: Construction of a Mapping

to the node n are replaced by a single edge from the intermediate node to the node n. Figure 7b illustrates a mapping of the network in Figure 7a where a node has been decomposed.

A two-level decomposition of a node can be represented by a set of groups $\mathcal{D} = \{d_1 \ldots d_g\}$. Each group d_i specifies either a single fanin node or a subset of fanin nodes covered by an intermediate node n_{d_i}. The minimum cost mapping of this decomposition is found by exhaustively searching all utilization divisions $\mathcal{U} = \{u_1 \ldots u_g\}$ and for each $\mathcal{U}$ constructing a minimum cost mapping of the decomposition where the root lookup table has utilization division $\mathcal{U}$. Since an intermediate node provides a single input to the root lookup table we add the requirement that $u_i = 1$ if the group d_i specifies an intermediate node.

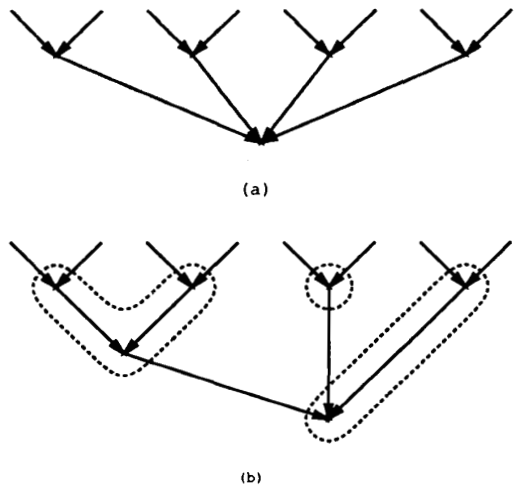

Figure 7: Decomposition of a Node

To proceed with the above search we require $minMap(n_{d_i}, K)$ for every possible intermediate node n_{d_i}. We find $minMap(n_{d_i}, K)$ by using an exhaustive search of utilization division of n_{d_i}.

It is also possible to have multi-level decompositions where an intermediate node n_{d_i} is itself decomposed. In this case we need to know $minMap(n_{d_j}, K)$ for all intermediate nodes n_{d_j} that cover a subset of the fanin nodes of n_{d_i}.

If we consider all possible intermediate nodes n_{d_i} in a sequence such that the number of fanin nodes covered by n_{d_i} increases from 2 to f then we can ensure that all $minMap(n_{d_j}, K)$ required to find any $minMap(n_{d_i}, K)$ have been previously calculated.

3.1.4 Node Splitting

As the fanin of a node increases the number of possible decompositions grows exponentially. However, the speed of our utilization division search and mapping construction makes it practical for us to consider all possible decompositions of a node as long as the fanin of the node is bounded by ten. For a node with fanin greater than ten the number of decompositions to be searched becomes impractically large. To reduce the execution time of Chortle we initially decompose such

large fanin node into two nodes with roughly equal fanin and then decompose each node separately.

This greatly reduces the number decompositions that need to be searched, but we can no longer guarantee finding the optimal decomposition. However, experimental results show that the mapping of a split node uses no more lookup tables than the mapping of the non-split nodes and are found in much less time. We believe the number of lookup tables is the same because for large fanin nodes there are many different minimum cost decompositions. As long as the chosen split does not preclude all of these decompositions Chortle will find one of the minimum cost decompositions.

4 Results

In this section the mapping quality and execution time of Chortle are compared with those of the MIS II technology mapping program [Detj87]. MIS II requires a library that represents every possible logic function that can be mapped. As discussed previously, a 4-input lookup table requires an impractically large library. The next section describes how we select a subset of all the possible functions to be represented in the library, for K greater than 3. The subsequent section gives the comparison.

4.1 Creating the MIS Library

The MIS library needs to contain only a single instance of all boolean functions that are permutations of each other. This reduces the number of elements required in the library to represent a K-input lookup table. For K=2 there are only 10 unique functions out of a possible 16, and for K=3 there are 78 unique functions out of a possible 256. We are therefore able to use complete libraries for K= 2 and 3. For K=4, there are a total of 9014 unique functions (out of a possible 65536), which is too large to represent in a MIS library.

The K=4 library was chosen by inspection of the library elements used by the K=3 results, and from a knowledge of the output of MIS. The logic optimization step in MIS finds a factored form for the network that minimizes the literal count. Such a network contains only level-0 kernels in the leaf nodes [Berg88]. The K=4 library thus includes the set of all level-0 kernels with four or fewer literals and their duals. This choice is supported by the observation that the MIS mapper used mostly level-0 kernels when mapping from the complete K=3 library. In addition, all level-n ($n > 0$) kernels that cannot be synthesized by level-0 kernels were included. A library constructed in this way includes all common circuit elements such as ANDs, AOIs and XORs.

We also greatly increase the *effective* coverage of the library because in the comparison below, we do not count the inverters used by MIS as logic blocks. This is because a simple post-processor could easily merge all inverters into the lookup tables. In doing this we give MIS credit for ability that it is not due – in several cases for K=3 (with the complete library), MIS was unable to select the correct version of a library cell with complemented inputs and had to use unnecessary inverters. The K=5 library is generated in a similar manner.

4.2 Experimental Results

To compare Chortle to MIS II we mapped several circuits from the MCNC-89 logic synthesis benchmarks for values of K from 2 to 5. The input networks for both mappers were optimized by the standard MIS II script. The results of these experiments are summarized in tables 1 to 4. Each table gives the benchmark name, the number of lookup tables mapped by the two programs, the % difference and the execution times on a SUN 3/60.

For K=2 the number of lookup tables in the Chortle and MIS mappings are nearly identical. This is expected because MIS is using a complete library and for K=2 all nodes will have to be completely decomposed into binary trees. Therefore the choice of decomposition does not matter. The four cases in which MIS achieves fewer lookup tables occur because the input network contains reconvergent fanout, such as XOR, which Chortle cannot find.

For K=3 the Chortle results average 6% better than the MIS results. Once again MIS is using a complete library so we would expect identical results. However with K=3 there is now the opportunity for the choice of decompositions to make a difference. Also the greedy algorithm used by MIS [Detj87] to deal with nodes with fanout greater than one tends to duplicate logic at fanout nodes. We have found that it is difficult to realize any savings by this greedy approach.

For K=4 the Chortle results average 9% better than the MIS results. In this case MIS is using an incomplete library and it is expected that its results would be worse.

For K=5 the Chortle results average 14% better than the MIS results. The lower coverage of the incomplete library for K=5 results in this increased difference.

The execution speed of Chortle ranges from a factor of 1 to 10 times faster than MIS II.

5 Conclusions

This paper describes an algorithm that effectively and efficiently performs technology mapping of boolean network into circuits of K-input lookup tables.

The encapsulation of mappings of nodes provided by the concepts of *utilization* and *utilization division* allows Chortle to reduce the search space thereby speeding the search for a minimum cost mapping. This fast search makes it practical for us to find the best decompositions of nodes as long as the fanin of the node is less

than or equal to ten. The algorithm will find the optimal solution when the input network is a tree. An experimental comparison to an adaptation of the MIS II mapper shows that significant gains are possible with this approach.

In the future we would like to address the problems of nodes with large fanin, reconvergent fanout within the network and optimizations that may result from the duplication of logic at fanout nodes. We would also like to extend our algorithm to handle commercial FPGA architectures.

Circuit	# tables MIS	# tables Chortle	%	t (sec.) MIS	t (sec.) Chortle
9symml	199	199	0.0	19.3	3.0
alu2	382	382	0.0	35.6	10.0
alu4	691	691	0.0	62.2	72.1
apex6	665	665	0.0	66.8	6.2
apex7	200	200	0.0	21.4	2.2
count	111	113	-1.8	12.8	0.9
des	3049	3049	0.0	299.2	82.2
frg1	111	111	0.0	11.4	0.9
frg2	737	740	-0.4	74.1	6.2
k2	811	811	0.0	78.0	23.9
pair	1439	1441	-0.1	134.6	22.7
rot	576	578	-0.3	56.0	5.5

Table 1: Results, K=2

Circuit	# tables MIS	# tables Chortle	%	t (sec.) MIS	t (sec.) Chortle
9symml	118	112	5.4	192.9	5.5
alu2	231	218	6.0	239.6	11.2
alu4	422	405	4.2	319.1	79.7
apex6	415	390	6.4	323.3	8.9
apex7	129	126	2.4	192.4	2.6
count	80	65	23.1	167.9	1.1
des	1866	1805	3.4	1016.2	176.0
frg1	64	60	6.7	167.4	1.2
frg2	481	452	6.4	339.0	7.0
k2	496	480	3.3	368.5	59.5
pair	917	851	7.8	524.4	45.8
rot	368	357	3.1	289.3	6.1

Table 2: Results, K=3

Circuit	# tables MIS	# tables Chortle	%	t (sec.) MIS	t (sec.) Chortle
9symml	86	78	10.3	21.9	7.9
alu2	177	159	11.3	78.7	12.4
alu4	314	286	9.8	68.6	86.2
apex6	273	261	4.6	71.4	11.5
apex7	98	94	4.3	23.9	3.1
count	63	49	28.6	15.8	1.2
des	1283	1225	4.7	298.6	266.3
frg1	47	43	9.3	14.3	1.3
frg2	346	333	3.9	80.7	7.7
k2	400	379	5.5	82.1	94.5
pair	669	635	5.4	137.4	67.9
rot	279	261	6.9	58.3	6.8

Table 3: Results, K=4

Circuit	# tables MIS	# tables Chortle	%	t (sec.) MIS	t (sec.) Chortle
9symml	74	63	17.5	45.1	10.2
alu2	157	131	19.8	72.0	13.4
alu4	291	238	22.3	119.1	93.2
apex6	251	234	7.3	122.5	14.4
apex7	86	73	17.8	46.9	3.7
count	47	47	0.0	32.5	1.3
des	1147	1075	6.7	390.8	353.3
frg1	41	34	20.6	31.4	1.6
frg2	325	278	16.9	134.4	8.9
k2	357	335	6.6	146.2	127.6
pair	574	504	13.9	178.8	88.7
rot	255	230	10.9	104.9	7.5

Table 4: Results, K=5

References

[Bake90] S. Baker, "AMD: Mach CMOS PLD a 'breakthrough'", Electronic Engineering Times, No. 581, March 12 1990 p. 8.

[Berg88] R.A. Bergamaschi, "Automatic Synthesis and Technology Mapping of Combinational Logic," Proc. ICCAD 88, Nov 1988, pp.466-469.

[Berk88] M. Berkelaar, J. Jess,"Technology Mapping for Standard Cell Generators", Proc. ICCAD 88, Nov 1988, pp. 470-473.

[Detj87] E.Detjens et. al, "Technology Mapping in MIS", Proc. ICCAD 87, Nov 1987, pp. 116-119.

[ElGa89] A. El Gamal, et. al, "An Architecture for Electrically Configurable Gate Arrays," IEEE JSSC Vol. 24, No. 2, April 1989, pp. 394-398.

[Fran91] R. J. Francis, "Ph.D. Thesis in preparation," University of Toronto, Department of Electrical Engineering.

[Hsie88] H. Hsieh, et. al "A 9000-Gate User-Programmable Gate Array," Proc. 1988 CICC, May 1988, pp. 15.3.1 - 15.3.7.

[Keut87] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," Proc. 24th Design Automation Conference, June 1987, pp. 341-347.

[Lisa87] R. Lisanke, F. Brglez, G. Kedem, "McMAP: A Fast Technology Mapping Procedure for Multi-Level Logic Synthesis," Proc. ICCD, pp. 252-256, October 1988.

[Marr89] C. Marr, "Logic Array Beats Development Time Blues," Electronic System Design Magazine, Nov. 1989, pp. 38-42.

[Ples89] Plessey Semiconductor ERA60100 preliminary data sheet.

[Rose89] J.S. Rose, R.J. Francis, P. Chow, and D. Lewis, "The Effect of Logic Block Complexity on Area of Programmable Gate Arrays," Proc. 1989 CICC, May 1989, pp. 5.3.1-5.3.5.

[Wong89] S.C. Wong, et. al, "A 5000-Gate CMOS EPLD with Multiple Logic and Interconnect Arrays," Proc. 1989 CICC, May 1989, pp. 5.8.1 - 5.8.4.